Proceedings of the 2007 Winter Simulation Conference
S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, eds.

HIERARCHICAL PLANNING AND MULTI-LEVEL SCHEDULING FOR SIMULATION-BASED PROBABILISTIC RISK ASSESSMENT

Hamed S. Nejad
Dongfeng Zhu
Ali Mosleh

Center for Risk and Reliability
University of Maryland
College Park, MD, 20742, U.S.A.

ABSTRACT

Simulation of dynamic complex systems—specifically, those comprised of large numbers of components with stochastic behaviors—for the purpose of probabilistic risk assessment faces challenges in every aspect of the problem. Scenario generation confronts many impediments, one being the problem of handling the large number of scenarios without compromising completeness. Probability estimation and consequence determination processes must also be performed under real world constraints on time and resources. In the approach outlined in this paper, hierarchical planning is utilized to generate a relatively small but complete group of risk scenarios to represent the unsafe behaviors of the system. Multi-level scheduling makes the probability estimation and consequence determination processes more efficient and affordable. The scenario generation and scheduling processes both benefit from an updating process that takes place after a number of simulation runs by fine-tuning the scheduler's level adjustment parameters and refining the planner's high level system model.

1 INTRODUCTION

Understanding the behavior of a dynamic system and assessing its performance under undesirable conditions and circumstances is a challenging task. In general, there are three main questions that need to be answered: 1) what can go wrong (scenarios); 2) what and how severe are the potential detriments (consequences); and 3) how likely are these undesirable consequences (probabilities). Simulation is often the only viable solution for this problem.

The good news is that many system designers and analysts are already using simulation for the design and virtual testing of their products. In other fields, simulation is used to model real world phenomena—such as the economy of a country, the behavior of different live organisms in a particular eco–system, or the people working in or interacting with an organization—to gain insight into the dynamics of the system. However, even simulation methods face a daunting task when confronting the abovementioned questions.

The number of events happening during a simulation run can be large if there are many details considered in the simulation model. Generating the risk scenarios becomes a major issue considering that the timing and order of the events can profoundly affect the outcome of the simulation. System resources can quickly become overwhelmed when the simulation tries to take on the task with all the details involved in the system element models. Observing the low probability- high consequence scenarios becomes a big challenge.

SimPRA, a simulation-based Dynamic Probabilistic Risk Assessment method is a framework accompanied by a prototype software tool developed at the University of Maryland to take on this challenge (Hu et al., 2004; Mosleh et al., 2004; Nejad and Mosleh, 2005; Zhu et al., 2006). A hierarchical planning engine generates high level risk scenarios automatically. Entropy-based biasing techniques are used to adaptively guide the simulation towards events and end-states defined in the plan. Multi-level scheduling is used to adjust the levels of detail of the simulation elements. The prior knowledge of the system and knowledge gained during simulation are used to dynamically adjust the exploration rules in the SimPRA environment.

The advantages of the SimPRA framework over other simulation-based techniques used for risk assessment can be grouped into the following four categories:

1.1 Handling a Large Number of Scenarios Without Compromising Completeness

The ability to handle a large number of scenarios without compromising completeness is one of the most challenging

problems that simulation-based risk assessment methodologies have encountered.

In discrete dynamic event tree simulations (Smidts, 1994; Labeaua et al., 2000), all combinations of events happening in a large number of discrete time steps define the universe of event sequence possibilities. A small system with five components that are either up or down (working or failed) observed in only 10 time steps will generate $\left(2^5\right)^{10}$ event sequences. Obviously some of these event sequences are either physically impossible or they are so low in probability compared to others that simulating them is not worth the effort

The classical way of dealing with this problem has been to define a certain number of criteria that will halt the progress of a scenario when the simulation encounters them:

1. Cut-off criteria: The probability of a scenario is constantly watched during the simulation run and when that probability falls lower than a certain level, that scenario will be considered to be an extremely rare scenario and will no longer be pursued.
2. Absorption point criteria: A set of rules are defined before running the simulation that would define the conditions under which the scenario will be deemed uninteresting. Examples are scenarios containing specific combinations of events or scenarios reaching specific pinch points.

A problem with these approaches is that sometimes a lot of precious simulation time is spent before reaching a cut-off or absorption point and the simulation is terminated without gaining any computational value out of it. Also, since these simulation methods employ search algorithms to find possible scenarios, searching in a large universe with a lot of uninteresting possibilities is a big waste of resources and can even overflow the system resources before achieving the results.

For continuous event tree simulation methods (Devooght and Smidts, 1999; Smidts, 1994; Labeaua et al., 2000), the primary challenge is in defining a *complete* set of risk scenarios. These methods generally rely on the natural probability of scenarios. Since most systems are designed to be highly reliable, the natural probability of a high-consequence scenario is likely to be very low. As such, a prohibitively large number of simulation runs may be required to ensure that those low-probability high-consequence scenarios are sufficiently explored. Recalling that risk is a function of consequences as well as probabilities, the generation of low-probability high-consequence scenarios is at least as important as the generation of high-probability low-consequence scenarios and the limitations of continuous event tree methods in this regard is significant.

The typical way of dealing with this problem has been the use of Importance Sampling. Importance Sampling artificially pushes the simulation to sample important or risk-relevant variables more frequently so that high-consequence scenarios are more likely to be explored. Simulation outputs are then weighted in order to try to correct for any bias that may have been introduced.

The SimPRA approach is a hybrid of the discrete dynamic event tree methods and continuous event tree methods that seeks to avoid the combinatorial explosion problem while simultaneously optimizing the completeness of the set of risk scenarios. Similar to the discrete methods, SimPRA begins by discretizing the world of possibilities into high-level events. Using hierarchical planning techniques and system behavioral knowledge, the world of combinatorial scenarios is reduced to a pool of physically, logically, and temporally possible high-level scenarios called a Plan. The simulation is run in a continuous event tree-type format that uses the Plan for biasing (Importance Sampling) purposes. Entropy rules with user-defined importance measures of events are used to select the branch points on an event tree generated from the Plan. Multi-level scheduling is also used to decide on the level of detail the events should have and to ensure that the simulation time is spent wisely on each detailed scenario generated by the simulator. Cut-off points and absorption points are also used to increase productivity.

1.2 Performing Risk Assessment Without Compromising the Complexity of The Problem

Most risk assessment tools use or provide a specific modeling framework that suits the theory behind their approach. The use of these modeling frameworks is not without cost:

1. The user should remodel the problem to fit the new framework which can be very time consuming. He or she also will need some risk assessment expertise.
2. The complexities in the model are often reduced to comply with the tacit and explicit assumptions behind the risk assessment methodology. This complexity reduction makes the system behavior model and risk assessment results increasingly unrealistic and hence decreases the accuracy or precision of the results.

The model used in SimPRA for risk assessment can be the same model that the designers have developed in the simulation environment to show how their system works. The simulation model might combine discrete and continuous events as the model for risk assessment. The only limitations on the simulation are the general computational limitations that a simulation environment might have and there is nothing imposed on it due to the risk assessment approach taken.

1.3 Increasing Simulation Efficiency by Adjusting the Level of Simulation Details Automatically

Most simulation tools generate detailed scenarios which include all different combinations of lowest level events. A lot of precious simulation time is wasted by simulating the unimportant components inside certain scenarios. The simulation model used in SimPRA is a multi-level model. The behavior of the component is abstracted at different levels. The simulation levels of detail are adjusted automatically during simulation. Multi-level abstraction is defined for selected components which meet the following criteria:

1. The simulation of the component is time-consuming (during simulation), and
2. The selected components are not important for certain scenarios.

The multi-level objects are simulated at a relatively high-level when they are not important to the scenario. The innovative aspect of the approach is that the selection of the most appropriate level of detail is initially specified by the analyst in the planner, but is then automatically adjusted during the various rounds of simulation according to an entropy-based rule.

1.4 Providing an Environment that Will Progressively Improve Efficiency Over Time

SimPRA provides a dynamic environment in which the risk assessment technique matures in the process. An updating algorithm suggests feedback to the analysts on how to update the planner's knowledge so that the planner will be able to generate observed scenarios that were not originally in the plan and stop generating high-level scenarios that do not correspond to any observed detailed scenarios generated by the simulator. After several refinements of the plan, plan-updating might stop giving new suggestions. This would be an ideal situation in which the plan is a reflection of all of the possible high-level scenarios generated by the simulator and none of the scenarios in the plan is left unexecuted.

2 SIMPRA FRAMEWORK OVERVIEW

The SimPRA framework is comprised of three major components: a *Simulator* that generates the detailed scenarios and identifies the ultimate fate of each one, a *Scheduler* which controls the timing and occurrence of the events and a *Planner* which is responsible for guiding the simulation through high level scenario generation. Figure 1 illustrates the main elements of the SimPRA framework as well as the interactions among them.

In the SimPRA framework, high level knowledge about the system's behavior and vulnerabilities is actively employed through a plan—or group of *high level* scenarios—to guide the simulator to generate those *detailed* scenarios that are most likely to generate normal and risky situations. These scenarios and their consequences, as identified by the simulator, are recorded and grouped for later study. The planner may 'learn' from the simulation results, and update the high level scenarios to better guide the scheduler in future runs of the simulation. Below, SimPRA components and the interactions among them are discussed in more detail.

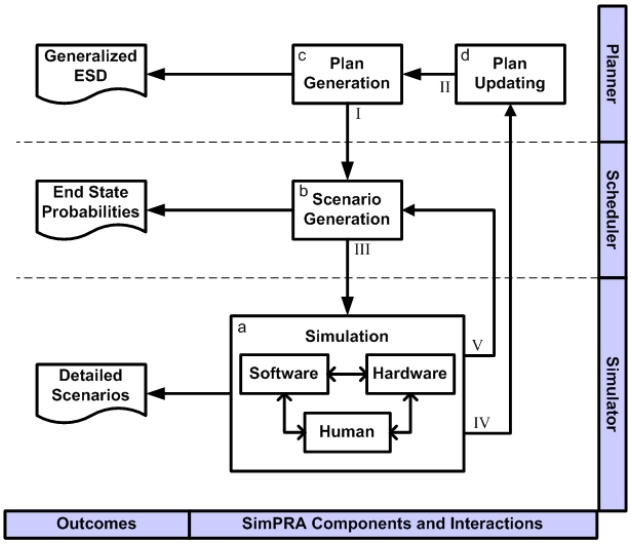

Figure 1: SimPRA framework components and the interactions among them

2.1 Simulator

The simulator is responsible for generating detailed behavior of the system as the system designers intended it to do. Simulation can consist of both discrete and continuous elements that would simulate the behavior of software, hardware or human components of a real system. It is even possible to let the real elements play a part in the simulation by providing an interface between them and the rest of the simulation program.

The granularity or the level of detail in a simulation model depends on many factors including the risk assessment goals, availability of data, computational power and the simulation speed.

There are two types of stochastic events in the simulator: time-based events and demand-based events. One type of stochastic behavior of a component can be described by a probability distribution function of time-to-failure. There is another class of failure. The probabilistic branching stochastic process has a set of outcomes, each with a probability of occurrence. The timing of the occurrence is not random; instead, the outcomes at that point of time are random.

To link the simulator with the scheduler, the simulation model needs to include some model elements that are able to notify the scheduler of the branch points and look

for guidance from it. Practically, this translates into using some elements from a tool box in the simulation model that are predefined to provide the communication. Most of these elements will be the elements with stochastic behavior that are guided by the scheduler.

Specifically, any simulation model in the Simulink® environment with embedded SimPRA tool-box components can be used by SimPRA for risk assessment. However, to make a simple model that only simulates the reliability behavior of the software, hardware and human elements of simulation without presenting other unnecessary aspects of these elements, several modeling techniques are developed and suggested for the SimPRA environment. Please refer to (Hu et al., 2004; Mosleh et al., 2004; Zhu et al., 2006) for more details.

In the case that multi-level objects are defined in hardware, software, or human components, a separate knowledge base needs to be constructed for each sub component. The prior knowledge related with level selection is stored in the knowledge base, including time-factor related information (the relationship between the high-level scheduler simulation time requirement and the simulation level of detail for this component) and the relationship between pre-defined component conditions and the simulation level of detail.

Data provided to the simulator (Figure 1: link III) contains the simulation commands that are executed in the branch points. These commands include: a) Checking some conditions in the simulation model; b) Setting some simulation variables that are directly provided by the scheduler and c) Running queries on the simulation model to decide the availability of options to set for the continuation of the simulation and perhaps doing a random selection among the possible options.

The simulator notifies the scheduler when branching points are generated. It also provides the results of the condition tests on the simulation elements when requested by the scheduler (Figure 1: link V). The simulator also keeps a record of all of the state changes in every simulation run with the time of these changes.

Figure 2 shows the algorithm behind the simulator component.

```
Set the simulation parameters
For each run of the simulation {
  Initiate event sequences
  While (not reached an end state or cut off time) {
    Perform continuous/discrete-time simulation
    If (Reached a branching point) {
      Inform the scheduler of the possible branches with their
            probabilities
      Wait for the scheduler to select a branch and for multi-level
          components select the level of detail by giving
          appropriate commands
      Execute the scheduler commands
    }
  }
  Inform the scheduler of reaching an end state or cut-off time
}
End the simulation
Report the results
```

Figure 2: SimPRA simulator algorithm

2.2 Scheduler

The scheduler manages the simulation process, by, among other things, saving system states, deciding the branch selection, adjusting the simulation levels of detail, and restarting the simulation. The scheduler guides the simulation toward the plan generated by the planner.

The scheduler keeps track of the simulation and guides it adaptively. The simulator proposes the transitions to the scheduler whenever the simulator reaches a branching point. The scheduler then retrieves the value of the proposed transitions and decides which branch is to be explored. The exploration command is sent to the simulator for execution and the simulation continues until the next branching point or an end state is reached. Two kinds of knowledge—prior knowledge and knowledge obtained from previous runs of the simulation—are used to guide the scenario exploration.

The prior knowledge is represented in the plan. In the SimPRA framework, the high level scenarios generated by the planner serve as a guide for the simulation (Figure 1: link I). The importance level based on prior information or engineering experience is integrated into the plan. The knowledge obtained during simulation is measured using expected entropy gain, which evaluates how much information is expected to be gained by simulating / exploring that scenario.

Shannon introduced the idea that information is a statistical concept and proposed using entropy as measure of information (Shannon, 1948). Lindley applied these ideas to measure the information in an experiment rather than in a message (Lindley, 1956). The amount of information provided by an experiment is measured by comparing the knowledge before and after the experiment, while the measure of information is given by Shannon's entropy. The details of the entropy-based exploration strategy are discussed in (Hu et al., 2004)

Due to the randomness of the simulation, it is believed that with a large number of guided simulation runs, all event sequences of interest would be generated. The guide is not expected to be complete, or even correct.

In addition to branch selection, the scheduler also handles the adjustment of the simulation levels in the case where multi-level objects are defined in the simulator. The planner is loaded into the scheduler at the beginning of the simulation. The simulation level of detail is adjusted adaptively based on the information in the plan, the information in the knowledge base of different components, and the previous simulation results.

A System Level Knowledge Base (SLKB) is established in the scheduler to represent the compatible combinations of the level of detail for different sub-components. The type of level control nodes includes:

1. Undefined: the level control is undefined.
2. Direct level control: the direct value for the level of simulation
3. Time Factor: the required time factor

The scheduler is established to handle multi-level objects in the simulation model if a multi-level plan is loaded. The scheduler checks the plan for level control commands and based on the type of the command it will take one of the following approaches:

1. If Level Control Node (LCN) contains the direct control information, the scheduler reads the level information from LCN and sets the simulation level of detail for all multi-level objects accordingly.
2. If LCN contains time factor requirements, the scheduler sends the time factor requirements to the sub-components. The sub-component queries its own knowledge-base to get the level control information based on the time factor requirement and sends it back to the high-level scheduler. The scheduler sets the simulation level of detail for all multi-level objects based on the combination.
3. If LCN is undefined, the scheduler sends this information to the sub-component. The sub-component decides the simulation level based on the information in its own knowledge base and sends it back to the high-level scheduler; the scheduler sets the simulation level of detail for all multi-level objects based on the combination.

After the start of simulation, the scheduler controls the simulation level of detail during each simulation run. The simulation continues until it reaches a LCN node. The simulation level of detail for multi-level objects only changes when the simulation reaches LCN. Figure 3 shows the algorithm behind the scheduler component.

It is necessary to mention that the term scheduler has been used in other simulation-based probabilistic risk assessment frameworks, such as DYLAM (Cojazzi, 1996) and ADS (Hsueh and Mosleh, 1996) as well. However, the role of the scheduler is quite different in our framework. In DDET implementations the scheduler directs the simulation to perform the systematic traversal of all the possible branches, typically in a depth-first manner. In the SimPRA framework, the scheduler is not only capable of performing the depth-first search as in the DDET, but is also able to adaptively guide the simulation toward the scenarios of interest by actively choosing branches. The scheduler's role is crucial since it has to be able to cover all the event sequence space, and it also has to maintain sufficient coverage of all of the planned scenarios and guide the simulation toward areas of greatest uncertainty. For the details of the scheduler algorithm please refer to (Hu et al., 2004; Zhu et al., 2006)

2.3 Planner

As mentioned earlier, the planner is responsible for guiding the simulation by providing the high-level scenarios to the scheduler. These scenarios might be incomplete or even in some cases incorrect. The planner itself, after a number of simulation runs, learns the detailed simulation results (Figure 1: link IV) and comes up with suggestions for the system analysts on ways to improve the plan model (Figure 1: link II) to generate unseen scenarios or eliminate the impossible ones.

```
Load the plan
Event tree ← Generate a tree of the plan data
Initialize multi-level objects
For each round of simulations {
  Reset the scenario values
  For each run of simulation {
    Weight of this simulation run = 1
    Wait for the simulator to send a message{
      If (end state message) {
        Add the weight of this simulation run to the end state weight
        Normalize weights to calculate the probabilities of end states

        End this simulation run
      }
      If (branching point message) {
        Find the node on the event tree
        Calculate the value of each possible branch
        Choose a branch based on the calculated values
        Calculate the weights needed to bias toward the branch
        Randomly choose a branch considering the weights
        Calculate the total weight of simulation run to this point
        If (weight is too high meaning reaching an absorption
            point){
          End this simulation run
        }
        Check if (a level control node is reached in the event tree){
      Get level control information
            if (the new level is different with the current level){
            Adjust the simulation level of detail
            }
        }
    }
  }
  Calculate Level Control Measure for each multi-level objects
  Update plan based on the calculation results
}
```

Figure 3: SimPRA scheduler algorithm

2.3.1 System Hierarchy

A structure tree shows the hierarchy of the system elements. Conceptually, a structure tree comes after the analysis of the system functionalities and the identification of how the functionalities are provided by different parts of the system. The system is usually supported by a number of components and subsystems. Subsystems themselves are comprised of additional groups of subsystems and components and so on.

2.3.2 State Transition Diagrams

In this step, information about the state transition graphs is obtained from the system modeler. Transition graphs show how each structure element provides the expected func-

tionalities through the changes or stays in different functional states.

A state of a system can, in general, be thought of as an optimal ensemble of system parameters which characterize it independent of its surroundings and history. When the system is in a state, its history of getting to that state is irrelevant to the future of its behavior.

Considering state transition graphs, it is possible to define three types of states: *source* states are the states that may be transitioned into; *sink* states are states that may be transitioned out of and *transient* states are those states that may be both transitioned into and out of.

The sink states of the *system*'s state transition diagram *only*, are considered as the end-states. All of the scenarios will end up in one of these end states. A state transition diagram should be a connected graph.Figure 4 shows an example of a state transition diagram. States are shown as the blue circles and transitions are depicted as boxes. Each transition can only have one input and one output state. The names of the transitions are given in the top of the transition boxes. It is possible to have several transitions connecting two states. It is also possible to see a transition name being used for several transitions but each should have a separate box with the same details. This means that there is the same underlying mechanism for providing those transitions.

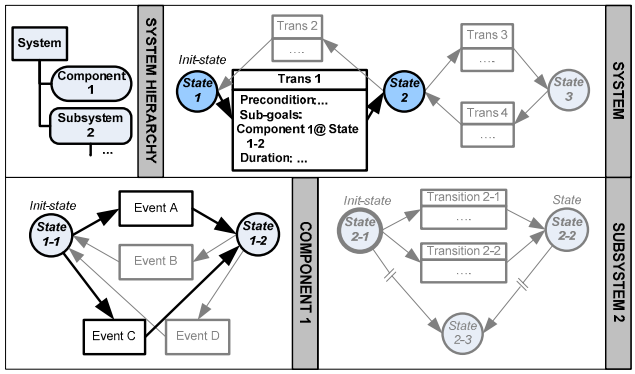

Figure 4: Hierarchical state transition diagrams of a generic system and its sub-elements

2.3.3 Transition Rules

Each of the transitions in the state transition diagram can be conditional on time, a landmark, or the state of another component of the system. If the condition is not met, it means the state of that particular element cannot be changed by the transition with the condition on it.

Each transition can also have a duration or set a landmark. This helps the modeler to condition the transitions on the clock of the system or by the scenario's achievements to a certain point in the scenario.

However, the most important thing is to define how the states of the system and subsystems are changed according to the states of their sub elements. This is where the hierarchical planning is shaped. The planner uses this knowledge to know how to change the state of sub elements to get the system and subsystems to a particular state. The specific rules defining and conditioning the transitions are as follows:

1. Duration: Each transition can have a duration time. When the planner reaches this rule, it adds the value of duration time to the time of the scenario. It's a good practice to keep the duration time for the lower level elements of the system.
2. Landmark: Each transition can set a landmark. When the planner reaches this rule, it adds the landmark mentioned in the command to the scenario.
3. Time condition: When the planner reaches a time condition, it compares the time of the scenario (which is the addition of duration times in the scenario to that point) with the time specified in the condition.
4. Landmark condition: When the planner evaluates a landmark condition, it searches in the scenario to see if the specified landmark is set before the condition or not.
5. State condition: The planner checks the state of the element referred to in the condition to see if the condition holds or not.
6. Sub-goals: This transition rule is only valid for the system and subsystem elements. The sub-elements of the system or subsystem and the state which can cause this transition are defined in this step. The planner considers these changes in the states to expand the plan to the lower level elements. The order of these transition rules are considered to be trivial to the process, however, the planner considers these transition rules in the order they appear. If the order can make a difference, another transition with the same transition rules but different order of elements can be added to the graph. The importance of each sub-goal is also set at this point.
7. Level control: This is just a note for scheduler to indicate if there is a need for switching the levels of detail during the simulation (see 2.2 for details)

2.3.4 Plan Generation

The planning process is done from top to bottom recursively. The planning goal that is taking the system from the initial-state to all (or sometimes a group of) end-states is considered first. All of the transition paths that take the system to its goals are determined. These paths will include all of the states and transitions that change these states to finally

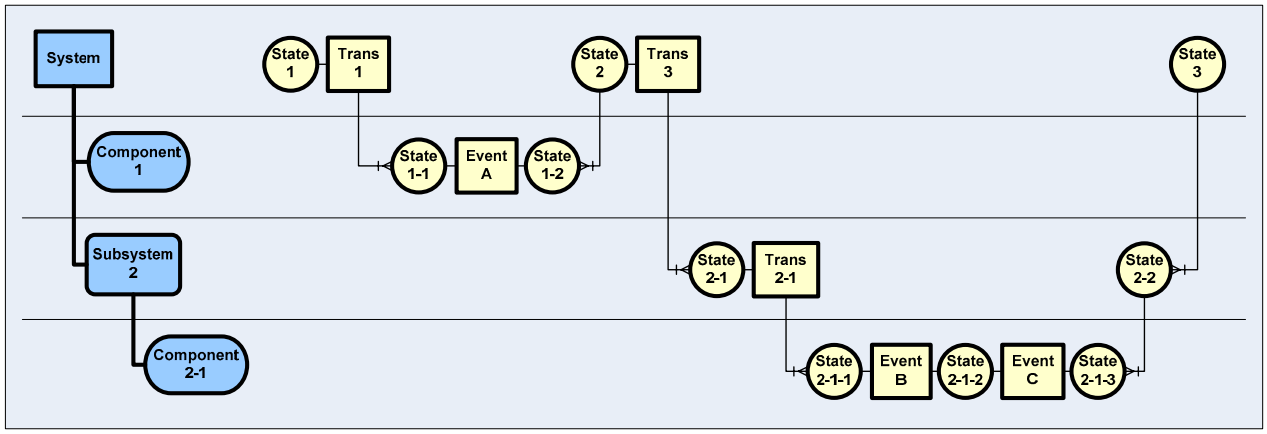

Figure 5: A high level scenario generated from the system hierarchy, state transitions and transition details

reach the goal state(s). The reason that state changes are included is the delay in the system response to the transitions and events during the simulation; the following transitions and events should only be called upon when the local states are reached. In other words, the presence of states works as a benchmark for the start of the next transition or event. Figure 5 depicts a plan generated from a generic state transition graph.

Obviously, for a transition to be considered in the plan, all of the preconditions of that transition need to be satisfied. Transitions that indicate the need for a multi level planning process (or need for satisfying a sub-goal) trigger the recursive planning process which in return will give all of the transitions and events that satisfy the sub-goals in the lower levels of system. Figure 6 provides the planner's algorithm.

2.3.5 Plan Updating

Updating starts when a predefined number of simulation runs are completed. The details of the scenarios generated by the simulator are used for two purposes: scenario updating and level-control updating.

For the planner, the updating engine checks every instance of a state change in the detailed scenarios for each element of the simulation. For components, if there is an event related to that component that is called between the state changes, that event will be considered as the cause of the state transition for that component. If there is no event between state changes, then the previous event will be considered as the source of change in the state of the component although that event is not recalled again in the simulation. Since components are the lowest level elements in the structure tree, there is no other element that can affect or describe their behavior. For subsystems, the last state of each of their subsystems or components is considered as the cause of the change in their states.

Transitions generated by the updating algorithms are compared with the state transitions of the system elements in the system model of the planner, and missing transitions will be suggested for addition while transitions not present in the updating data will be suggested for elimination. There are obviously other constraints on the transitions that the updating algorithm is not capable of determining.

The second purpose of updating is to adjust the level-control information in the plan. The post simulation analysis uses a level control measure, which is a combination of Shannon's entropy measure (Lindley, 1956; Shannon, 1948; Shannon and Weaver, 1949) and a risk measure.

The Shannon entropy value is calculated for each multi-level object in the plan. The risk measure can be:

1. The conditional probability for the end state of interest after the multi-level object node. This measure focuses on the consequence of the system when the multi-level objects fail. The conditional probability for the end state of interest is a value ranging from 0 to 1. A large value indicates that the failure of this multi-level object has a large probability of leading the simulation to the end state of interest.
2. The importance measure of the multi-level objects in the system. This measure focuses on the importance of the multi-level objects to the end state of interest.

Birnbaum (Birnbaum, 1969) introduced the concept of importance. The measure of component *i* is described as:

$$I_i^B = \frac{\partial R_s[r_i(t)]}{\partial r_i(t)}$$

Where

I_i^B = Birnbaum importance of component *i*

$R_s[r_i(t)]$ = Reliability of the system as a function of the reliability of the individual component

$r_i(t)$ = Reliability of component *i*

In the case of simulation, $R_s[r_i(t)]$ is replaced by the probability of the end state of interest $P_e[r_i(t)]$. $r_i(t)$ represents the probability of failure of the multi-level component i.

In the case that the multi-level object only has two different states: fail or success, the Birnbaum importance measure can be calculated as:

$$I_i^B = P_e[r_i(t)=1] - P_e[r_i(t)=0]$$

A large value of the importance measure indicates that the status of this component influences the end state of interest dramatically. Thus the multi-level objects should be further decomposed to get more accurate results. A small value of the importance measure indicates that the status of this component does not influence the end state of interest. Thus the multi-level object should be simulated at a relatively high level.

The threshold for the level control measure is defined by the analyst before the simulation begins. It can be further updated after each round of simulation. After each round of simulation, the level control measure for each multi-level object is calculated and compared with the level information in the previous LCN. If the value of the level control measure indicates that the simulation level should be adjusted, the scheduler submits a request to the multi-level objects to check if any updates are available. If no further updates are available for the multi-level objects, the plan is treated as the best plan available. The simulation results are calculated based on this plan.

3 CONCLUSION AND LESSONS LEARNED

This paper presented the SimPRA framework which incorporates hierarchical planning and multi-level scheduling in order to better guide the simulation of complex systems for the purposes of probabilistic risk assessment. During the process of modeling several systems for plan generation (Hu et al., 2006; Mosleh et al., 2004; Zhu et al., 2005), the following lessons have been learned that make the modeling process easier and smoother:

- It is better to keep details in the component level and make the subsystem level elements as light as possible. This is a very important concept since it can make the modifications to the model much easier and make the planner more expandable.
- If there seems to be a component that belongs to several subsystems, it is likely that the component should be considered as an independent subsystem. An example of this is the power system which usually provides service to most of the subsystems. There is a tendency to consider it as an element of all subsystems, but in fact, it is better to be considered as an independent subsystem.
- It is better to keep the model as similar to the real system as possible instead of making assumptions to reduce the size of the model. For example, if the real system has 4 layers of subsystems to get to the components, it is better to keep the model the same instead of merging some of the subsystems into each other.

```
Plans ← empty
Element ← first subsystem
Path ← System transition paths from initial-state to goal states
state* ← System's initial-state
Get goal* and revisit-number
If (Element's state* equals Element's goal*) return Plans
Paths ← an empty set of lists of events {( )}
(*) For each Path-Line of Path{
        Candidate-Events ← Element Events that their Start-State
              equals the End-State of the last event of the Path-Line
              unless the Start-State will be revisited more than re-
              visit-number. If Path-Line has no event or last event
              does not belong to the element then add events that
              their Start-State equals state*
        New-Paths ← {}
        ∑ ← Run the Path-Line to get the latest states of the elements
              of the world
        For each Candidate-event in Candidate-events{
            Remove the Candidate-event if its pre-conditions do not
                  hold based on ∑
            Make a copy of Path-Line and add the Candidate-event
                  to the end of the list and add it to New-Paths
            For each New-Path in New-Path Set Element to element
                  of the last event and Plan recursively}
        Replace the Path-Line with New-Paths}
    Add Path-Lines that their last event's End-State equals goal* to Plan
          and remove them from Paths
    If Paths and New-Paths are not empty go to (*)
```

Figure 6: High level planning process algorithm

ACKNOWLEDGEMENTS

This work has been supported by a grant from NASA Ames Research Center, Engineering for Complex Systems.

REFERENCES

Birnbaum, Z. W., 1969, *On the importance of different components in a multicomponent system*: Multivariate Analysis 11, P.R.Krishnaiah, ed.,Academic Press.

Cojazzi, G., 1996, The DYLAM approach for the dynamic reliability analysis of systems. *Reliability Engineering and System Safety* 52:279-296.

Devooght, J., and C. Smidts. 1999. Probabilistic reactor dynamics. The theory of continuous event trees. *Nuclear Science and Engineering* 111:229–40.

Hsueh, K. S., and A. Mosleh. 1996. The development and application of the accident dynamic simulator for dynamic probabilistic risk assessment of nuclear power plants. *Reliability Engineering and System Safety* 52:297-314.

Hu, Y., F. Groen, and A. Mosleh. 2004. An Entropy-Based Exploration Strategy in Dynamic PRA. International Conference on Probabilistic Safety Assessment and Management (PSAM7).

Hu, Y., H. Nejad, D. Zhu, and A. Mosleh. 2006. Solution of the Phased-Mission Benchmark Problem Using the SimPRA Dynamic PRA Approach. International Conference on Probabilistic Safety Assessment and Management (PSAM) 2006.

Labeaua, P.E., C. Smidts, and S. Swaminathan. 2000. Dynamic reliability: towards an integrated platform for probabilistic risk assessment. *Reliability Engineering and System Safety* 68:219–254

Lindley, D. V. 1956. On the Measure of Information Provided by an Experiment. *Annals of Statistics* 27:986-1005.

Mosleh, A., F. Groen, Y. Hu, H. Nejad, D. Zhu, and T. Piers. 2004. Simulation-Based Probabilistic Risk Analysis Report, Center for Risk and Reliability, University of Maryland.

Nejad, H., and A. Mosleh. 2005. Automated Risk Scenario Generation Using System Functional and Structural Knowledge. In *Proceedings of ASME International Mechanical Engineering Congress and Exposition*.

Shannon, C. E. 1948. Mathematical theory of communication. *The Bell Labs Technical Journal* 379-457.

Shannon, C. E., and W. Weaver, 1949, *The mathematical theory of communication*. Urbana,, IL: University of Illinois Press.

Smidts, C. 1994. Probabilistic dynamics: a comparison between continuous event trees and a discrete event tree model. *Reliability Engineering and System Safety* 44:189–206.

Zhu, D., A. Mosleh, and C. Smidts. 2005. Software Modeling Framework for Dynamic PRA. *European Safety & Reliability Conference 2005* 2099-2107.

Zhu, D., A. Mosleh, and C. Smidts. 2006. A Framework to Integrate Software Behavior into Dynamic Probabilistic Risk Assessment. *Reliability Engineering and System Safety* (forthcoming).

AUTHOR BIOGRAPHIES

HAMED S. NEJAD is a Ph.D. student of Reliability Engineering at the University of Maryland. His current research interest is simulation-based probabilistic risk assessment of complex systems comprised of hardware, software and human elements. He is also a risk and reliability engineer for ELORET Corp. He can be contacted directly at <h.s.nejad@gmail.com> or by mail at NASA Ames Research Center, MS# 258-1, Moffett Field, CA 94035-1000.

DONGFENG ZHU is a faculty research associate in the Reliability program of the Mechanical Engineering Department at the University of Maryland. Dr. Zhu's research interest lies in the area of Probabilistic Risk Assessment, Risk Analysis, Software Reliability, Human Reliability, and Financial Risk. His homepage can be found via <http://www.glue.umd.edu/~zhudf> and his email address is <zhudf@umd.edu>.

ALI MOSLEH is a professor in the Mechanical Engineering Department at the University of Maryland. Dr. Mosleh has made research contributions in Bayesian Methods for Data Analysis; Methodology for Use of Expert Quantitative Opinion, Reliability Growth Modeling, Probabilistic Reliability Physics, Common Cause Failure Analysis, Modeling the Impact of Organizational Factors on System Reliability, Dynamic Accident Simulation and Dynamic Probabilistic Risk Assessment, Human Reliability Analysis (Cognitive Modeling, Simulation and Data Classification), Methodology for Information Systems Security Risk Management, and Space Systems Risk Analysis. He is an expert consultant to national and international organizations on risk and reliability issues. His web page can be found via <http://www.enme.umd.edu/> and his Email address is <mosleh@umd.edu>.